CHALMERS UNIVERSITY OF TECHNOLOGY | UNIVERSITY OF GOTHENBURG

Course on Computer Communication and Networks

Lecture 2-cont
Chapter 2 (part a): applications, http

EDA344/DIT 420, CTH/GU

Based on the book Computer Networking: A Top Down Approach, Jim Kurose, Keith Ross, Addison-Wesley.

Chapter 2: Application Layer

Chapter goals:

- conceptual + implementation aspects of network application protocols
  - client server, p2p paradigms (*we will study the latter seperately*)
  - service models
- learn about protocols by examining basic application-level protocols (more will come later, when studying real-time traffic aspects)

- specific protocols:
  - http, (ftp), smtp, pop, dns, p2p file sharing
- programming network applications
  - socket programming

Roadmap

- Applications and their needs, vs Internet trasport layer services
- The http protcocol
  - General description and functionality
  - Authentication, cookies and related aspects
  - Caching andp roxies
- (continuation with more applications: next lecture)

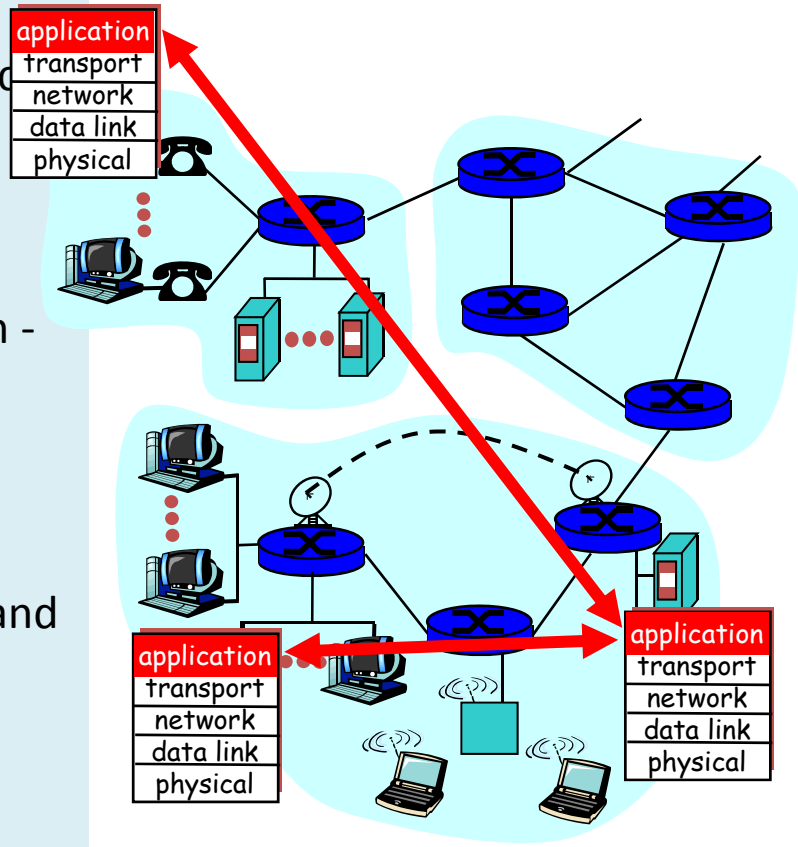
Applications and application-layer protocols

Application: communicating, distributed processes

- running in network hosts in “user spac
- exchange messages
- e.g., email, file transfer, the Web

Application-layer **protocols**

- Are only one “piece” of an application - others are e.g. user agents.
  - Web:browser
  - E-mail: mail reader
  - streaming audio/video: media player
- **define** messages exchanged by apps and actions taken
- **use services** provided by lower layer protocols



Client-server paradigm

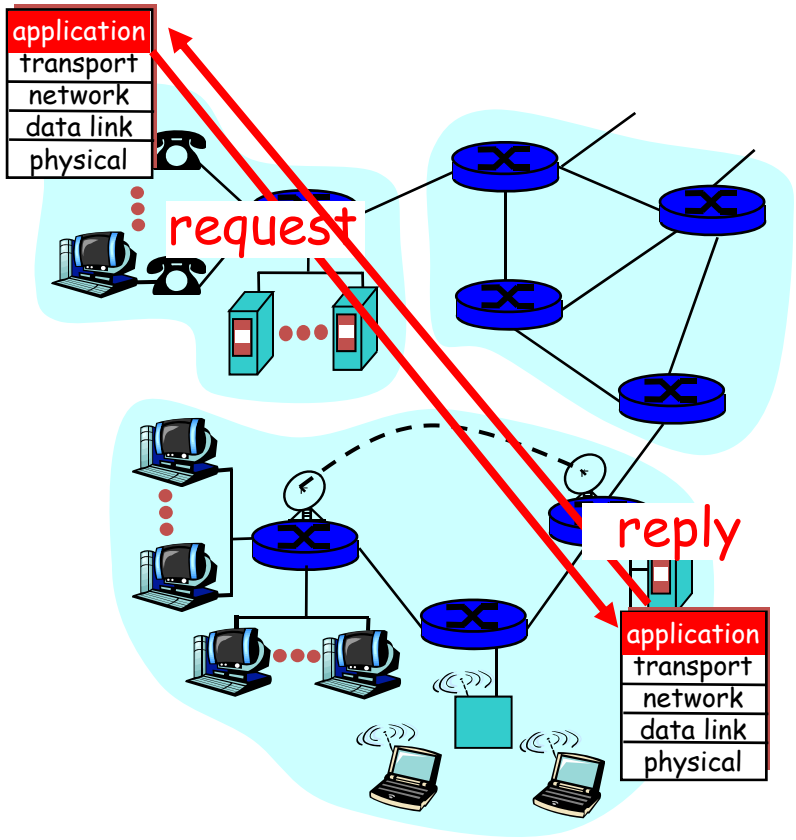
Typical network app has two pieces:
client and *server*

Client:

- initiates contact with server ("speaks first")
- typically requests service from server,
- for Web, client is implemented in browser; for e-mail, in mail reader

Server:

- provides requested service to client
- e.g., Web server sends requested Web page, mail server delivers e-mail



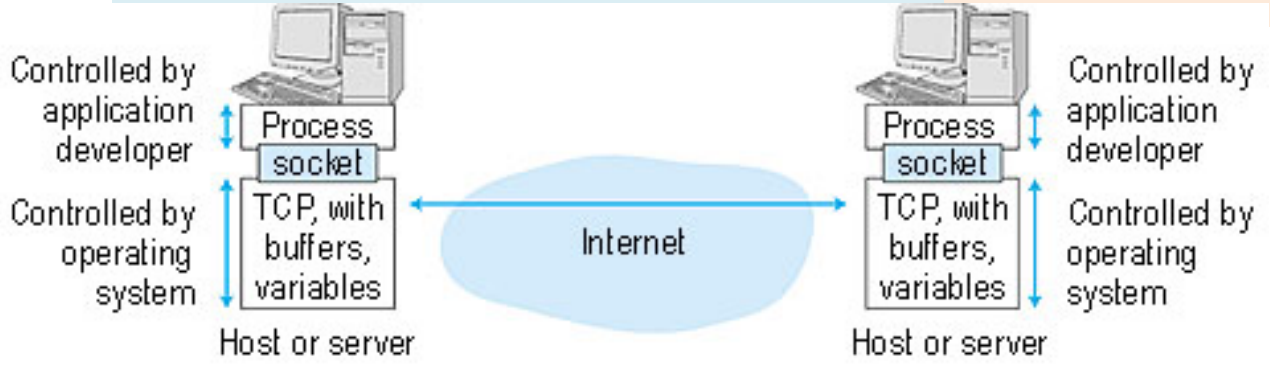
Auxiliary terms ++

socket: Internet application programming interface

- 2 processes communicate by sending data into socket, reading data out of socket (like sending out, receiving in via doors)

Q: how does a process "identify" the other process with which it wants to communicate?

- IP address of host running other process
- "port number" - allows receiving host to determine to which local process the message should be delivered



… more: later, with TCP/UDP + cf programming project guidelines

Properties of transport service of interest to the app

Data loss

- some apps (e.g., audio) can tolerate some loss
- other apps (e.g., file transfer, telnet) require 100% reliable data transfer
- Connection-oriented vs connectionless services

Bandwidth, Timing, Security

- some apps (e.g., multimedia) require minimum amount of bandwidth
- some apps (e.g., Internet telephony, interactive games) require low delay and/or low jitter
- other apps (elastic apps, e.g. file transfer) make use of whatever bandwidth, timing they get
- some apps also require confidentiality and integrity (more in network security)

Transport service requirements of common apps

Application	Data loss	Bandwidth	Time Sensitive
file transfer	no loss	elastic	no
e-mail	no loss	elastic	no
Web documents	no-loss	elastic	no
real-time audio/video	loss-tolerant	audio: 5Kb-1Mb video:10Kb-5Mb	yes, 100's msec
stored audio/video	loss-tolerant	rather similar	yes, few secs
interactive games	loss-tolerant	few Kbps up	yes, 100's msec
financial apps	no loss	elastic	yes and no

Services by Internet transport protocols

TCP service:

- *connection-oriented:* setup required between client, server
- *reliable transport* between sending and receiving process

+ a bit more, to focus in the TCP study

- *flow control:* sender won't overwhelm receiver
- *congestion control:* throttle sender when network overloaded
- *does not provide:* timing, minimum bandwidth guarantees

UDP service:

- *connectionless*
- *unreliable transport* between sending and receiving process

- *does not provide*: flow control, congestion control, timing, or bandwidth guarantee

Q: why bother? Why is there a UDP?

Internet apps: their protocols

Application	Application layer protocol	Underlying transport protocol
e-mail	» smtp [RFC 821]	TCP
remote terminal access	telnet [RFC 854]	TCP
Web	» http [RFC 2068]	TCP
file transfer	ftp [RFC 959]	TCP
streaming multimedia	proprietary (e.g. RealNetworks)	TCP or UDP
remote file server	NSF	TCP or UDP
Internet telephony	SIP, RTP, proprietary (e.g., Skype)	typically UDP, TCP also possible
nslookup and many others	» DNS [RFC 882, 883,1034,1035]	UDP

Roadmap

- Applications and their needs, vs Internet trasport layer services
- The http protcocol
  - General description and functionality
  - Authentication, cookies and related aspects
  - Caching andp roxies
- (continuation with more applications: next lecture)

The Web: some jargon

- Web page:
  - consists of "objects"
  - addressed by a URL
- Most Web pages consist of:
  - base HTML page, and
  - several referenced objects.
- URL has two components: host name and path name:

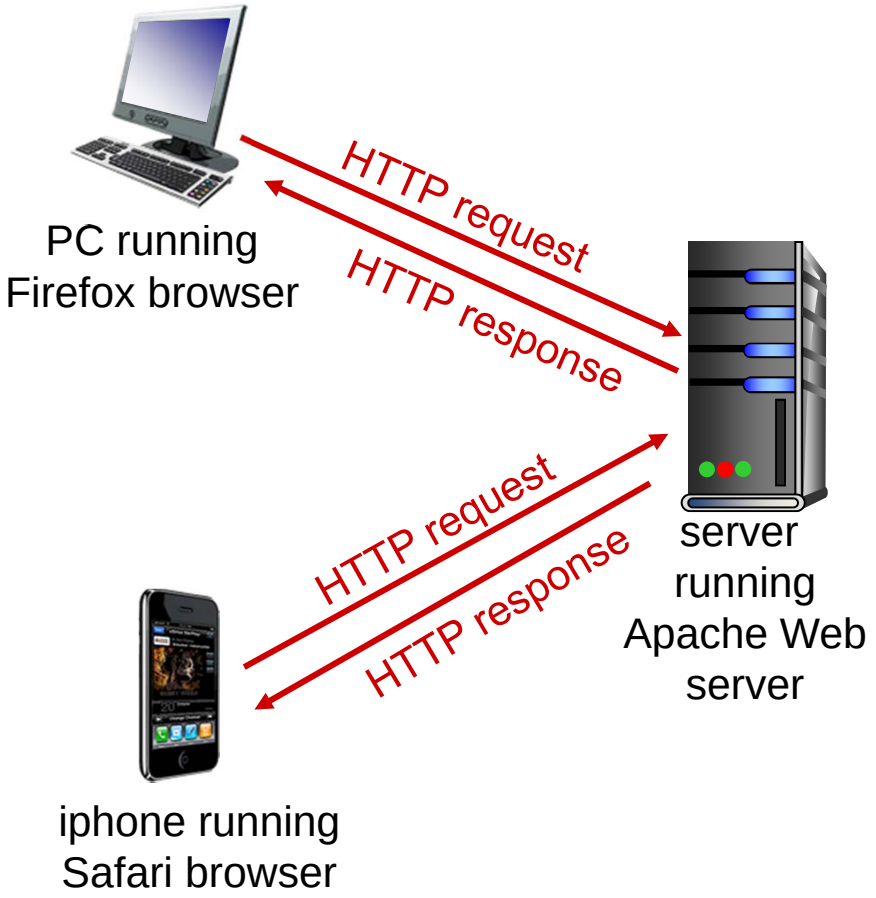
www.someSchool.edu/someDept/pic.gif

- User agent for Web is called a browser:
  - MS Internet Explorer
  - Netscape Communicator
- Server for Web is called Web server:
  - Apache (public domain)
  - MS Internet Information Server
  - Netscape Enterprise Server

HTTP overview

HTTP: hypertext transfer protocol

- Web's application layer protocol
- client/server model
  - *client:* browser that requests, receives, (using HTTP protocol) and "displays" Web objects
  - *server:* Web server sends (using HTTP protocol) objects in response to requests



HTTP overview (continued)

uses TCP:

- client initiates TCP connection (creates socket) to server, port 80
- server accepts TCP connection from client
- HTTP messages (application-layer protocol messages) exchanged between browser (HTTP client) and Web server (HTTP server)
- TCP connection closed

HTTP is "stateless"

- server maintains no information about past client requests

aside

protocols that maintain "state" are complex!

- past history (state) must be maintained
- if server/client crashes, their views of "state" may be inconsistent, must be reconciled

http example

Suppose user enters URL

www.someSchool.edu/someDepartment/home.index

(contains text, references to 10 jpeg images)

1a. http client initiates TCP connection to http server (process) at www.someSchool.edu. Port 80 is default for http server.

1b. http server at host www.someSchool.edu waiting for TCP connection at port 80. "accepts" connection, notifying client

2. http client sends http *request message* (containing URL) into TCP connection socket

3. http server receives request message, forms *response message* containing requested object (someDepartment/home.index), sends message into socket

time

http example (cont.)

4. http server closes TCP connection.

5. http client receives response message containing html file, displays html. Parsing html file, finds 10 referenced jpeg objects

6. Steps 1-5 repeated for each of 10 jpeg objects

time

Non-persistent and persistent connections

Non-persistent

- HTTP/1.0
- server parses request, responds, and closes TCP connection
- new TCP connection for each object => extra overhead per object

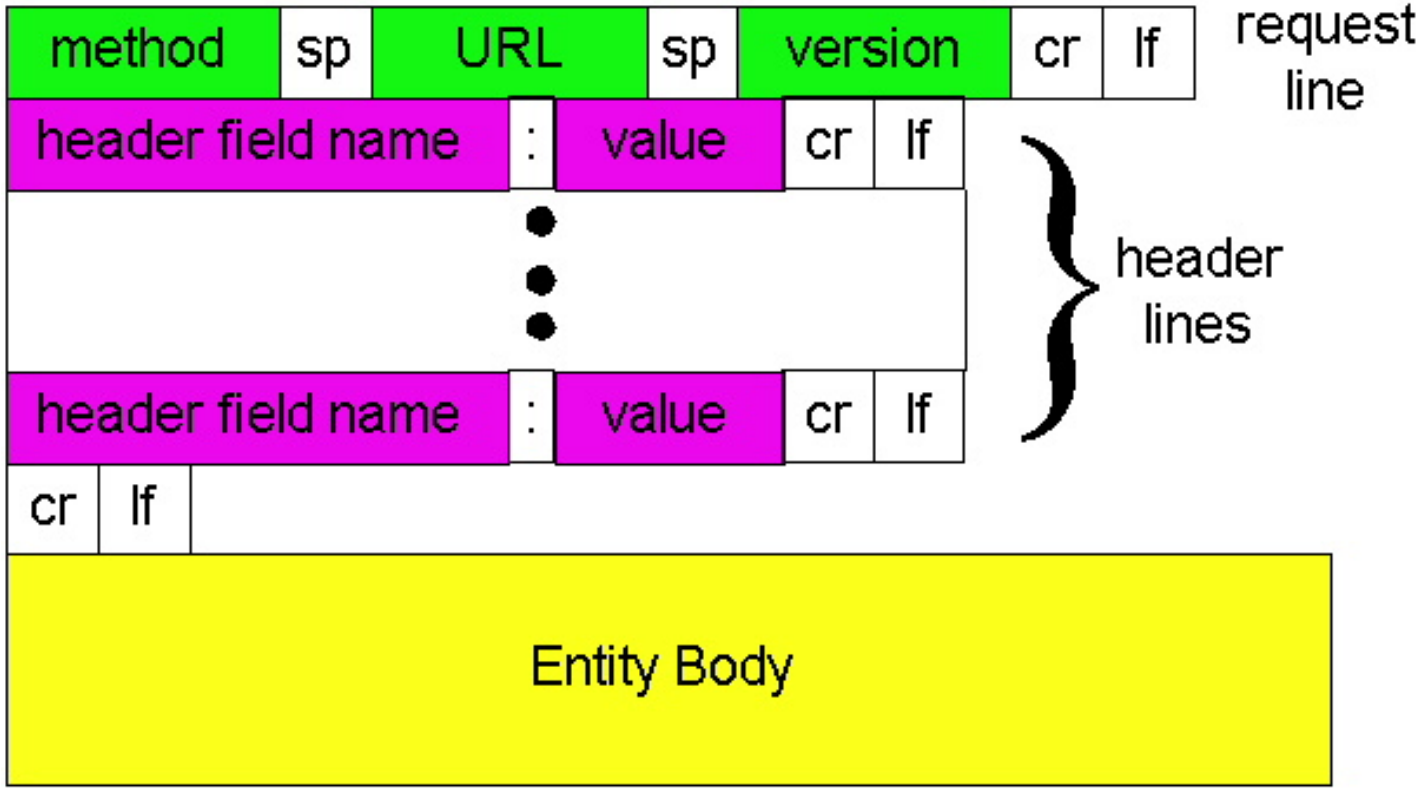
Persistent

- default for HTTP/1.1
- on same TCP connection: server parses request, responds, parses new request,..
- Client sends requests for all referenced objects as soon as it receives base HTML;
- Less overhead per object
- Objects are fetched sequentially

But most 1.0 browsers use parallel TCP connections.

But can also pipeline requests, to parallelize (resembles non-persistent optimised behaviour)

http request message: general format



HTTP request message

- two types of HTTP messages: *request, response*
- HTTP request message:
  - ASCII (human-readable format)

Method types

HTTP/1.0:

- GET
- POST
- HEAD
  - asks server to leave requested object out of response

HTTP/1.1:

- GET, POST, HEAD
- PUT
  - uploads file in entity body to path specified in URL field
- DELETE
  - deletes file specified in the URL field

HTTP response message

status line (protocol status code status phrase)

header lines

data, e.g., requested HTML file

```
HTTP/1.1 200 OK\r\n
Date: Sun, 26 Sep 2010 20:09:20 GMT\r\n
Server: Apache/2.0.52 (CentOS)\r\n
Last-Modified: Tue, 30 Oct 2007 17:00:02 GMT\r\n
ETag: "17dc6-a5c-bf716880"\r\n
Accept-Ranges: bytes\r\n
Content-Length: 2652\r\n
Keep-Alive: timeout=10, max=100\r\n
Connection: Keep-Alive\r\n
Content-Type: text/html; charset=ISO-8859-1\r\n
\r\n

data data data data data ...
```

For more headers:
www.w3.org/Protocols/HTTP/1.1/draft-ietf-http-v11-spec-01.html

http response status codes

In first line in server->client response message.

A few sample codes:

200 OK

- request succeeded, requested object later in this message

301 Moved Permanently

- requested object moved, new location specified later in this message (Location:)

400 Bad Request

- request message not understood by server

404 Not Found

- requested document not found on this server

505 HTTP Version Not Supported

Trying out HTTP (client side) for yourself

1. Telnet to your favorite Web server:

```
telnet cis.poly.edu 80
```

opens TCP connection to port 80 (default HTTP server port) at cis.poly.edu. anything typed in sent to port 80 at cis.poly.edu

2. type in a GET HTTP request:

```
GET /~ross/ HTTP/1.1
Host: cis.poly.edu
```

by typing this in (hit carriage return twice), you send this minimal (but complete) GET request to HTTP server

3. look at response message sent by HTTP server!

(or use Wireshark to look at captured HTTP request/response)

Roadmap

- Applications and their needs, vs Internet trasport layer services
- The http protcocol
  - General description and functionality
  - Authentication, cookies and related aspects
  - Caching andp roxies
- (continuation with more applications: next lecture)

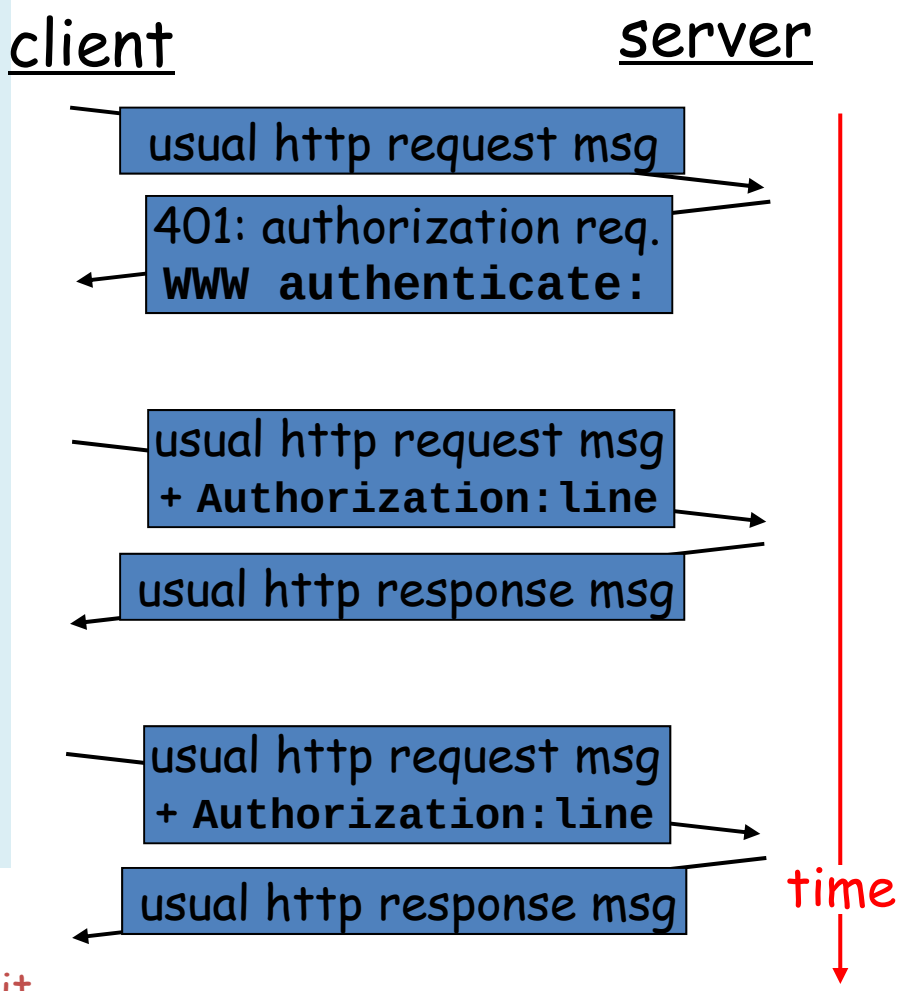
User-server interaction: authentication

Authentication goal: control access to server documents

- http is stateless: client must present authorization in each request
- authorization: typically name, password
  - **`authorization:`** header line in request
  - if no authorization presented, server refuses access, sends

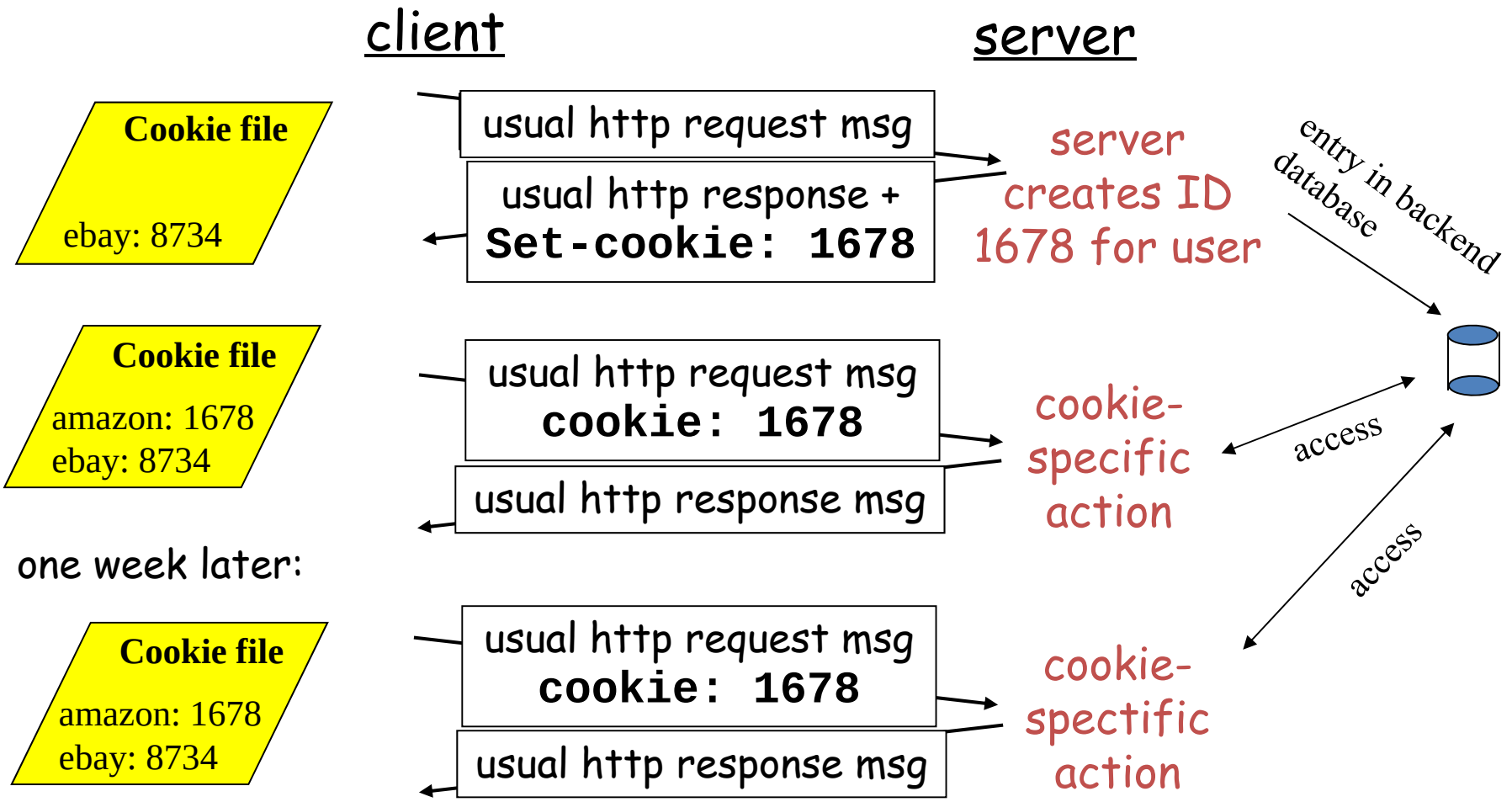
    `WWW authenticate:`

    header line in response



Browser caches name & password so that user does not have to repeatedly enter it.

Cookies: keeping "state"



Cookies (continued)

cookies can bring:

- authorization
- shopping carts
- recommendations
- user session state

aside

Cookies and privacy:

- cookies permit sites to learn a lot about you
- you may supply name and e-mail to sites
- search engines use cookies to learn yet more
- advertising companies obtain info across sites

Roadmap

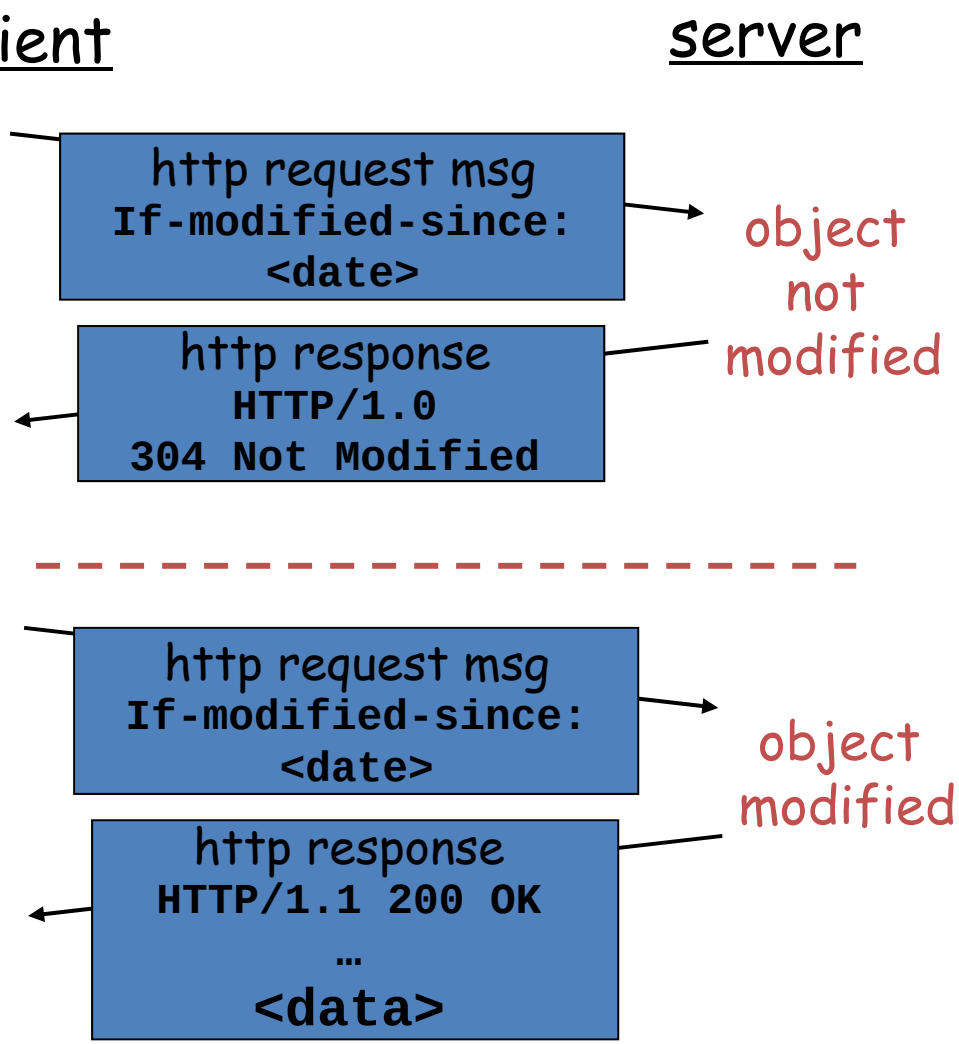
- Applications and their needs, vs Internet trasport layer services
- The http protcocol
  - General description and functionality
  - Authentication, cookies and related aspects
  - **Caching and proxies**
- (continuation with more applications: next lecture)

Conditional GET: client-side caching

- Goal: don't send object if client has up-to-date stored (cached) version
- client: specify date of cached copy in http request

  If-modified-since: <date>
- server: response contains no object if cached copy up-to-date:

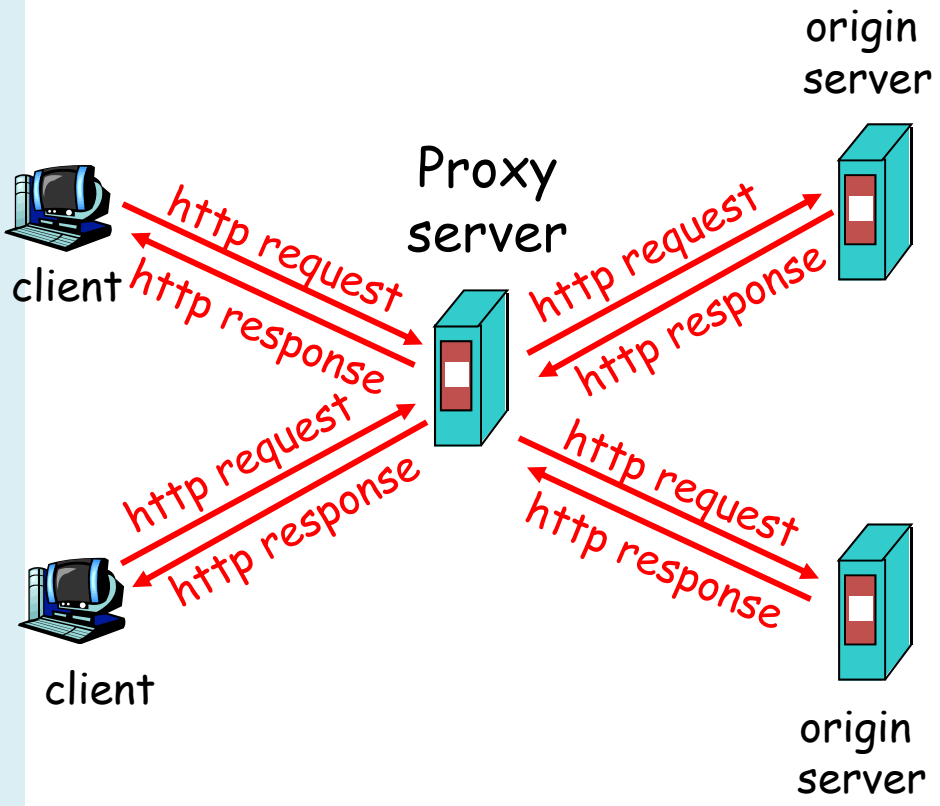
  HTTP/1.0 304 Not Modified



Web Caches (proxy server)

Goal: satisfy client request without involving origin server

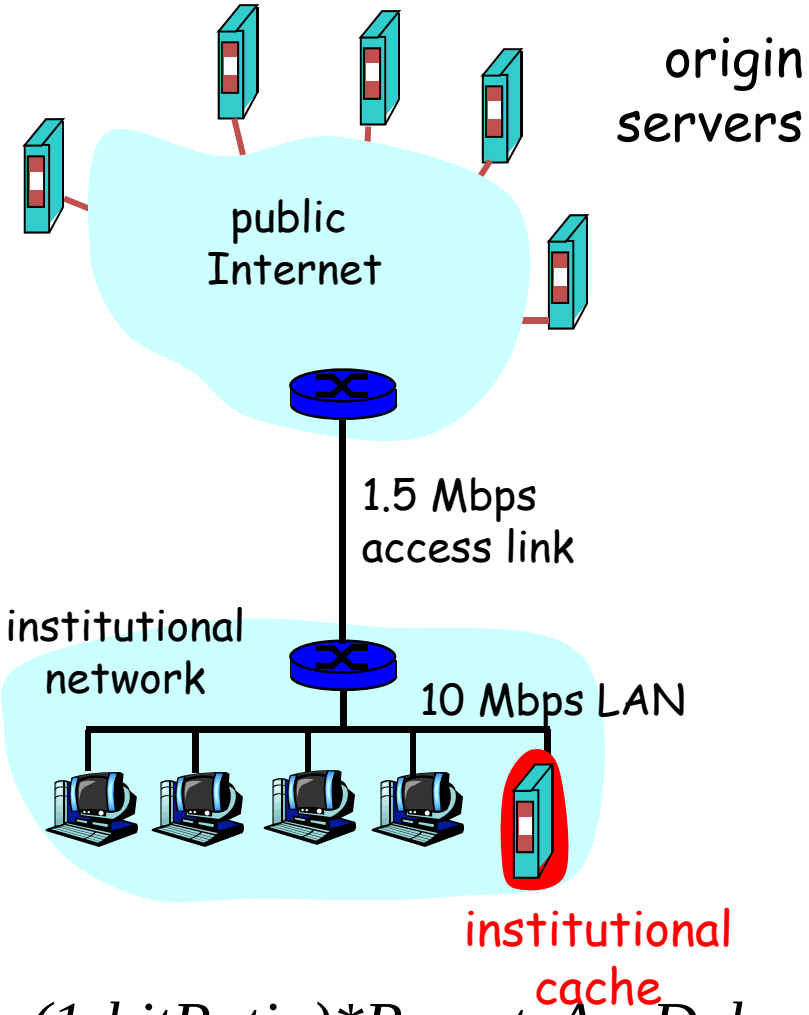
- user configures browser: Web accesses via web cache
- client sends all http requests to web cache; the cache(proxy) server
  - if object at web cache, return object in http response
  - else request object from origin server (or from next cache), then return http response to client
- Hierarchical, cooperative caching, ICP: Internet Caching Protocol (RFC2187)



Why Web Caching?

Assume: cache is "close" to client (e.g., in same network)

- smaller response time: cache "closer" to client
- decrease traffic to distant servers
  - link out of institutional/local ISP network often bottleneck
- Important for large data applications (e.g. video,…)
- Performance effect:



$$E(delay)=hitRatio*LocalAccDelay + (1-hitRatio)*RemoteAccDelay$$

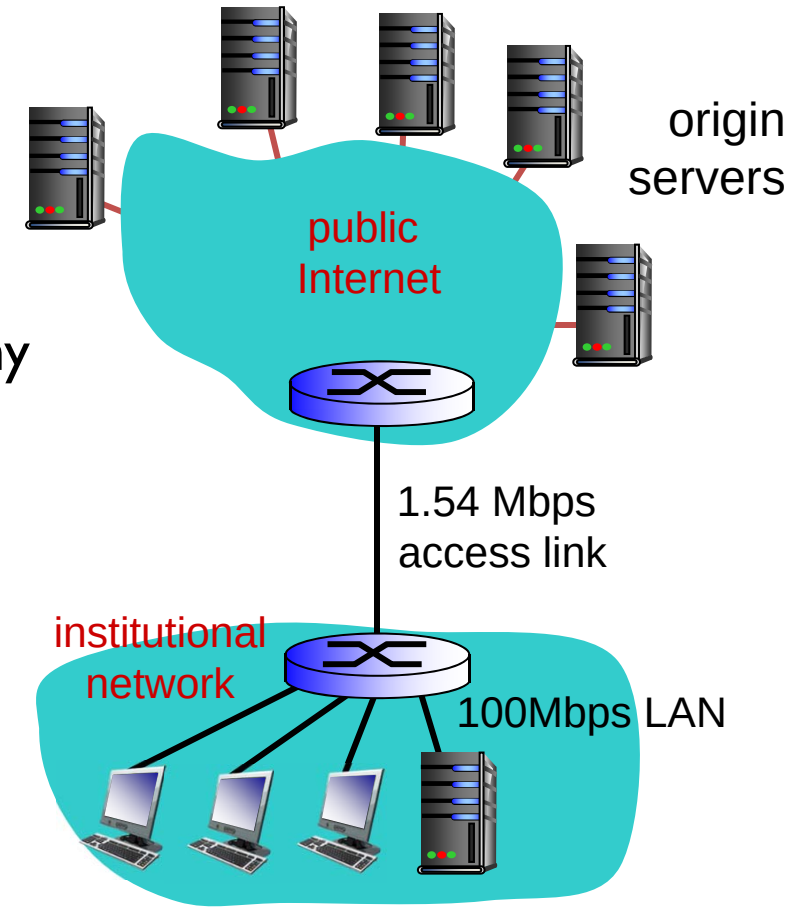
Caching example:

assumptions:

- avg object size: 100K bits
- avg request rate from browsers to origin servers:15/sec
  - i.e. avg data rate to browsers: 1.50 Mbps
- RTT from institutional router to any origin server: 2 sec
- access link rate: 1.54 Mbps

consequences:

- LAN utilization: 1.5% *problem!*
- access link utilization = 99%
- total delay = Internet delay + access delay + LAN delay
  = 2 sec + minutes + quite_small



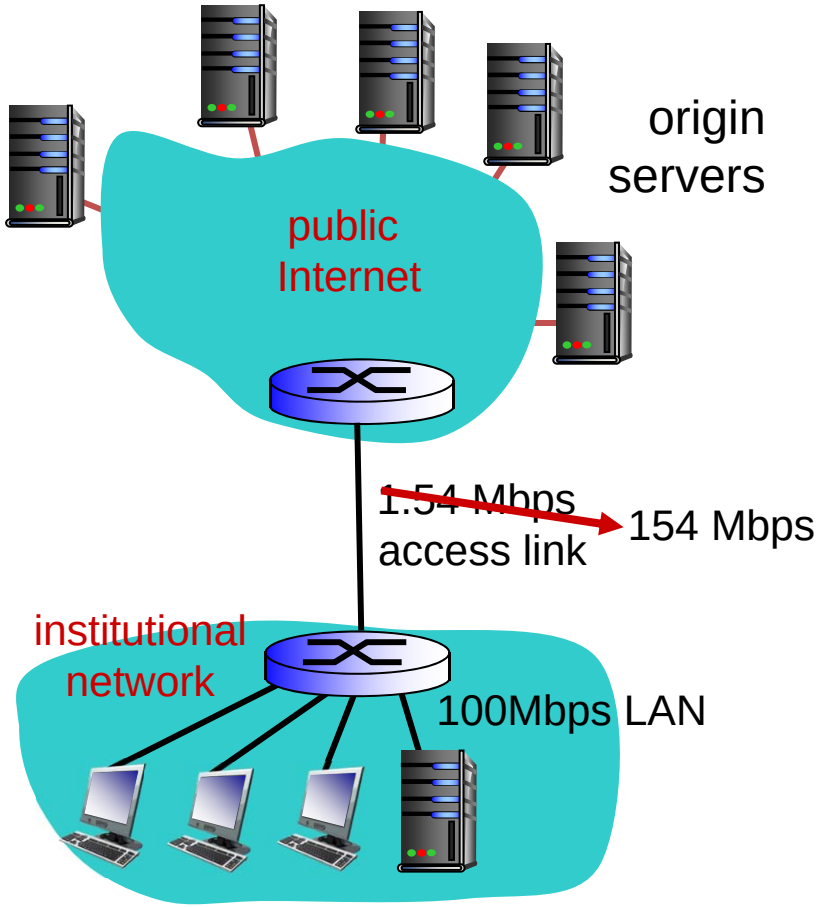
Caching example: faster access link

assumptions:

- avg object size: 100K bits
- avg request rate from browsers to origin servers: 15/sec
  - i.e. avg data rate to browsers: 1.50 Mbps
- RTT from institutional router to any origin server: 2 sec
- access link rate: ~~1.54 Mbps~~ 154 Mbps

consequences:

- LAN utilization: 1.5%
- access link utilization = ~~99%~~ 9.9%
- total delay = Internet delay + access delay + LAN delay
  = 2 sec + minutes + usecs ~~msecs~~



Cost: increased access link speed (not cheap!)

Caching example: install local cache

assumptions:

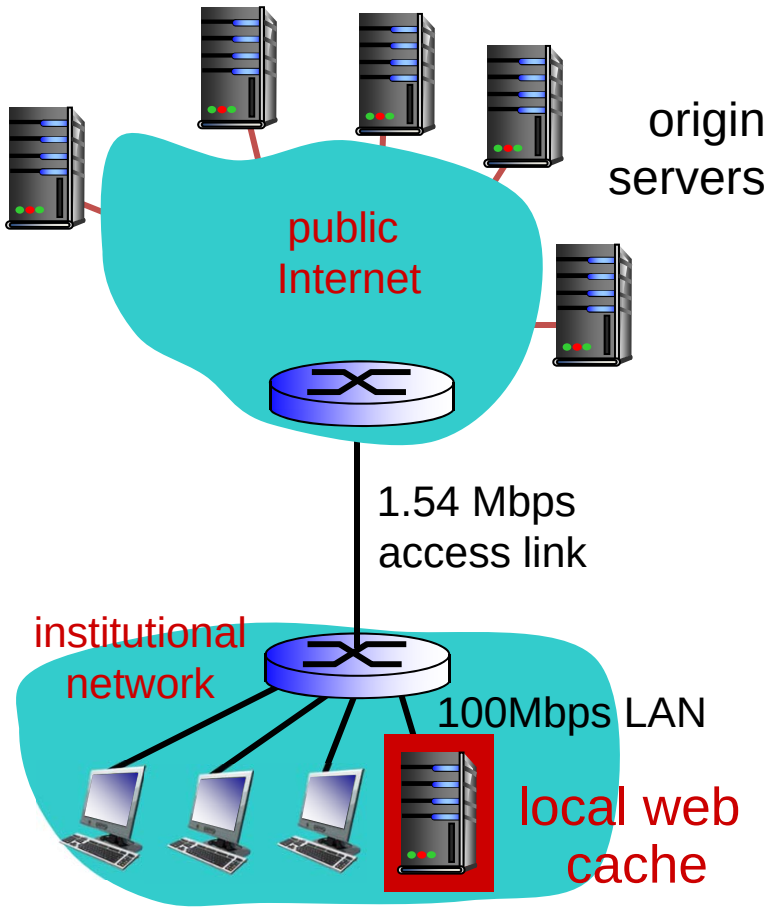
- avg object size: 100K bits
- avg request rate from browsers to origin servers: 15/sec
  - i.e. avg data rate to browsers: 1.50 Mbps
- RTT from institutional router to any origin server: 2 sec
- access link rate: 1.54 Mbps

consequences:

- LAN utilization: 1.5%
- access link utilization ?
- total delay ?

How to compute link utilization, delay?

Cost: web cache (cheap!)



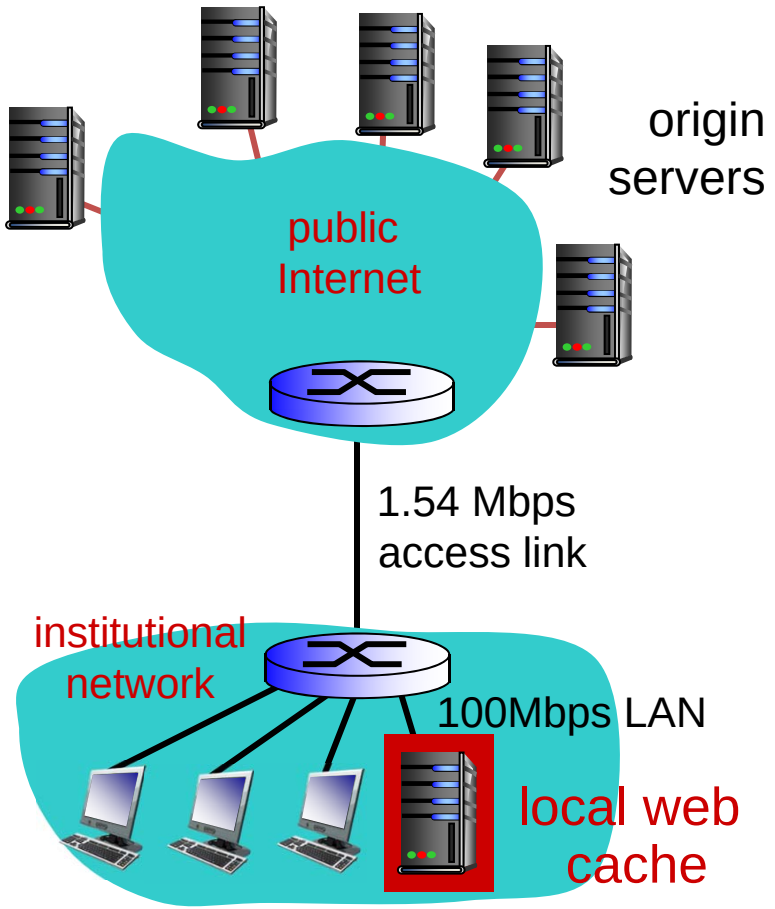
Caching example: install local cache

Calculating access link utilization, delay with cache:

- suppose cache hit rate is 0.4
  - 40% requests satisfied at cache, 60% requests satisfied at origin

- access link utilization:
  - 60% of requests use access link
- data rate to browsers over access link = 0.6*1.50 Mbps = .9 Mbps
  - utilization = 0.9/1.54 = .58

- total delay
  - = 0.6 * (delay from origin servers) +0.4 * (delay when satisfied at cache)
  - = 0.6 (2.01) + 0.4 (~msecs)
  - = ~ 1.2 secs
  - less than with 154 Mbps link (and cheaper too!)



Roadmap

- Applications and their needs, vs Internet trasport layer services
- The http protcocol
  - General description and functionality
  - Authentication, cookies and related aspects
  - Caching andp roxies
- (continuation with more applications: next lecture)